A Quantitative Approach to Assessing System Evolvability

John A. Christian, III*
NASA Johnson Space Center, Houston, TX 77058

ABSTRACT

When selecting a system from multiple candidates, the customer seeks the one that best meets his or her needs. Recently the desire for evolvable systems has become more important and engineers are striving to develop systems that accommodate this need. In response to this search for evolvability, we present a historical perspective on evolvability, propose a refined definition of evolvability, and develop a quantitative method for measuring this property. We address this quantitative methodology from both a theoretical and practical perspective. This quantitative model is then applied to the problem of evolving a lunar mission to a Mars mission as a case study.

1 INTRODUCTION

Evolvability, the ability of a system to adapt to changing requirements, is a property many engineers seek when developing a new system. Unfortunately, the system properties that result in evolvability are not well understood, making it hard to measure a system's potential to evolve and even harder to build an evolvable system. We approach this problem from the ground up, and begin by asking ourselves two questions:

1) "What do we want in an evolvable system?"
2) "Why do we want an evolvable system?"

1.1 Background- Historical Perspective and Current Events

Our story begins in the early 1970s as the waterfall model is introduced by W. Royce [Buede, 2000]. This model was originally developed for the software engineering development process but has been adapted to systems engineering. Characterized by sequential life-cycle phases, the waterfall model is well known throughout the software and systems engineering communities. Unfortunately, the waterfall model only allows iteration between neighboring phases and does not capture iteration between widely separated phases (something that is not uncommon in the development process). The waterfall model formed the basis for DoD-STD-2167A Defense Systems Software Development [Buede, 2000] which was released in February 1988. During the late 1980s and early 1990s this document was the default standard in the United States and throughout the world. While DoD-STD-2167A never explicitly dictated the waterfall model, it was perceived by many to do so. This model, at times, could be restrictive and prevent developers from following the most efficient development strategy [Newberry, 1995]. In response to these concerns, DoD-STD-2167A was replaced by MIL-STD-498 in December 1994. MIL-STD-498 was an interim document intended to increase flexibility until a commercial equivalent could be developed. This happened four years later with the release of IEEE/EIA 12207 in April 1998 (at this point MIL-STD-498 was canceled and replaced by IEEE/EIA 12207).

In the late 1980s B. Boehm created a different software development model: the spiral model [Boehm, 1988]. This model accounted for iterations and evolution not found in the waterfall model. As with the waterfall model, the spiral model began its life in software engineering and later made the transition to systems engineering. About the same time Boehm authored the spiral model, K. Forsberg and H. Mooz introduced the Vee model, often referred to as incremental development [Mooz and Forsberg, 2004; Buede, 2000]. Recently, a strong preference for spiral and evolutionary development (as seen in the spiral and vee models) has been stated by the Department of Defense (DoD) [DoD Directive 5000.1, 2003; DoD Instruction 5000.2, 2003] and recommended to the National Aeronautics Space Administration (NASA) [Aldridge et al., 2004].

* Phone number: (770) 330-9732; Facsimile number: (404) 894-2760; e-mail: gte230w@mail.gatech.edu

1.2 Evolutionary Acquisition and Spiral/Incremental Development

DoD Instruction 5000.2 [2003] states that "evolutionary acquisition is the preferred DoD strategy for the rapid acquisition of mature technology for the user. An evolutionary approach delivers capability in increments, recognizing, up front, the need for future capability improvements. The objective is to balance needs and available capability with resources, and to put capability into the hands of the user quickly." Simply put, evolutionary acquisition is the "successive improvement of solution versions based on experience with prior versions" [Mooz and Forsberg, 2004]. The two preferred models for achieving evolutionary acquisition are spiral and incremental development. DoD Instruction 5000.2 [2003] defines these approaches as follows:

Spiral Development: In this process, a desired capability is identified, but the end-state requirements are not known at program initiation. Those requirements are refined through demonstration and risk management; there is continuous user feedback; and each increment provides the user the best possible capability. The requirements for future increments depend on feedback from the users and technology maturation.
Incremental Development: In this process, a desired capability is identified, an end-state requirement is known, and that requirement is met over time by developing several increments, each dependent on available mature technology.

The spiral model is shown in Figure 1. In this model, the radial dimension represents the cumulative cost and the angular dimension represents the progress made in completing each cycle of the spiral [Boehm, 1988]. The model is characterized by four major processes (clockwise from top left of spiral): design, evaluation and risk analysis, development and testing, and planning. These processes may be repeated as many times as necessary [Buede, 2000]. The spiral model, however, is not perfect. Just as with the waterfall model, there are cases where the spiral model can be restrictive. Indeed, Mooz and Forsberg [2004] caution against using evolutionary development or the spiral model in cases where they would not provide the most effective approach. What is important to note here is that there is probably not a one-model-fits-all solution.

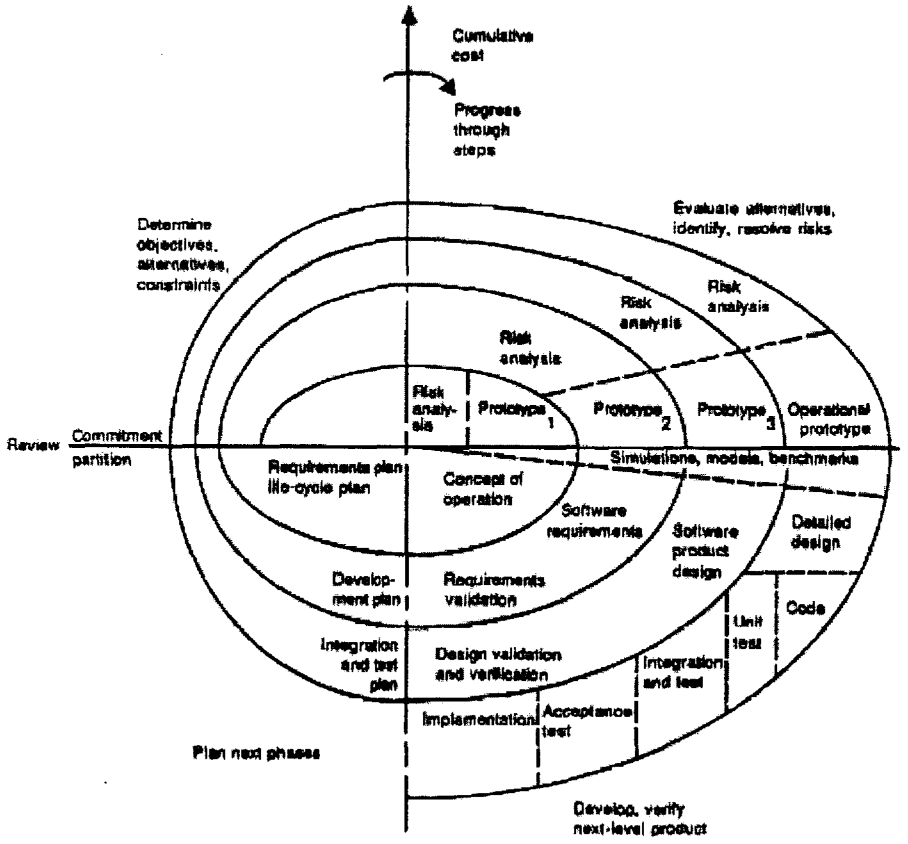


Figure 1. Spiral model [Boehm, 1988].

1.3 Why Measures of System Evolvability Are Needed

We now return to the two questions posed at the beginning of this introduction. In response to these, we conclude the following:

1) In an evolvable system, we look for a system that can adapt to new requirements at a cost less than what is required to build a new system.
2) We want an evolvable system because it gives the customer additional flexibility. With an evolvable system the customer can use existing infrastructure to tackle changing requirements- both planned and unplanned.

With these objectives in mind, we begin our search for a quantitative measure of system evolvability. It is clear that evolutionary development requires an evolvable system to be effective. Therefore when comparing multiple systems, and intending to use evolutionary development, it is important to compare and contrast their potential to evolve. Such a comparison lends itself to a qualitative comparison through a narrative, pros/cons list, or similar means. As is discussed in more detail later, comparing systems in a numerical fashion is preferable to the sometimes easier qualitative descriptions. In order to develop a quantitative measure for system evolvability, we must first create a clearer definition of this property.

2 DEFINING EVOLVABILITY

Now that we have established what we want in an evolvable system and why we want it, we proceed by developing a more precise definition of evolvability. Over the years, there have been many attempts to define evolvability [Isaac and McConaughy, 1994; Percivall, 1994; Rowe and Leaney, 1997; Rowe, Leaney, and Lowe, 1998]. Based on these past definitions, we propose the following definition of evolvability:

Evolvability: The capacity of a system to successfully adapt to changing requirements [IEEE, 1990] throughout its lifecycle without compromising architectural integrity. Furthermore, an evolvable system must meet the new needs of the customer in a more cost effective manner than developing a new system.

Despite a rather straightforward definition, evolvability can take many different forms. After all, there are many ways that a system can accommodate change. It can be argued, therefore, that the larger property of evolvability is actually a composite of the many methods a system possesses for adapting to change. Understanding this functional breakdown, termed the "taxonomy of change" by Rowe, Leaney, and Lowe [1998], is fundamental to developing a quantitative measure of the larger property of evolvability. Generally, evolvability may be subdivided into two categories: static and dynamic (or robust and flexible using the terminology of Saleh, Hastings, and Newman [2001]). Further inspection shows that a system can adapt to change in one of four ways: make no changes to the system (generality), reconfiguring existing system components (adaptability), increasing the size of existing system components (scalability), or adding new components (extensibility). Figure 2 graphically illustrates the functional breakdown of evolvability.

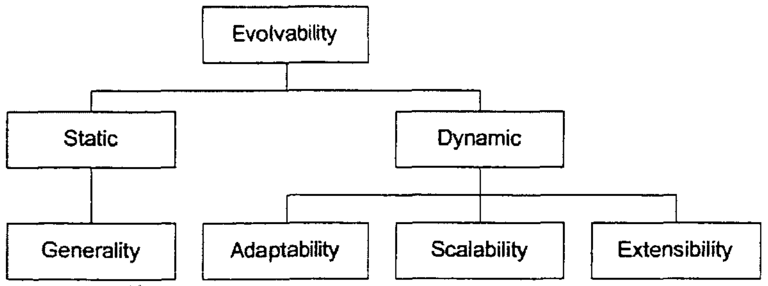


Figure 2. A functional breakdown of evolvability.

Detailed definitions of the four classes of evolvability, largely inspired by the work of Rowe, Leaney, and Lowe [1998] and information available in the SMC Systems Engineering Primer and Handbook [2004], are as follows:

Generality: The capacity of a system to accommodate a change in requirements without altering the existing architectural design or implementation strategy.
Adaptability: The capacity of a system to accommodate a change in requirements through rearranging existing system components within the current architecture without changing other components or their integration solution.
Scalability: The capacity of a system to accommodate a change in requirements by increasing the size of architectural components to accommodate increased loads.
Extensibility: The capacity of a system to accommodate a change in requirements through adding new components or through a major change in the architecture or implementation strategy.

3 FIGURES OF MERIT

The first step when evaluating a system is to consult the customer to identify their needs and objectives. Then figures of merit (FOM) that reflect these desires may be selected. A FOM is an objective and quantitative measure of system effectiveness. According to the NASA Systems Engineering Handbook [1995: 83], "a measure of system effectiveness describes the accomplishment of the system's goals and objectives quantitatively. Each system (or family of systems with identical goals and objectives) has its own measure of system effectiveness. There is no universal measure of effectiveness for NASA systems, and no natural units with which to express effectiveness." Simply put, FOMs help the analyst and the customer identify the system that best meets their objectives.

A good FOM should be worded and calculated such that a higher value represents a more desirable result. In addition, according to the SMC Systems Engineering Primer and Handbook [2004], a good FOM also has the following seven characteristics:

1) Relates to performance
2) Simple to state
3) Complete
4) States any time dependency
5) States any environmental conditions
6) Can be measured quantitatively (if required, may be measured statistically or as a probability)
7) Easy to measure

Of particular importance to the work that follows is the sixth characteristic- the ability to quantitatively assess a FOM. A numerical value is important, among other reasons, for comparison and compilation purposes. It is difficult to rank a number of qualitative descriptions, such as narratives, while ordering a set of numbers is straightforward. This problem is compounded when there is a need to compare aggregate FOMs.

3.1 Standard Scoring Functions

Earlier, we introduced the four classes of evolvability: generality, adaptability, scalability, and extensibility. From this description, it is clear that evolvability is an aggregate quantity, making it difficult to measure this property directly. Instead, we measure each class of evolvability and then combine them to achieve an overall measure evolvability. Each class of evolvability may be viewed as a bottom-level FOM, while the overall property of evolvability may be viewed as a top-level FOM.

Because the four bottom-level FOMs will be combined, it is necessary to standardize and normalize their output. Standard scoring functions, as described by Daniels, Werner, and Bahill [2001], are used to do this. These mathematical functions accept parameters specific to a FOM as an input, and yield a real number on a defined scale (usually between zero and one). Such functions are useful because they can take in a variety of parameters (with different units and different scales) for different FOMs and transform these inputs into a set of values on a scale common to all FOMs. These scoring function curves can have a number of shapes, but a higher FOM score should indicate a more desirable outcome.

The scoring functions used here are the 12 families of standard scoring functions introduced by A.W. Wymore [1993] and are identified by SSFx, where x is the family of scoring function. These functions restrict the output on a scale between zero and one. The nomenclature used is as follows (from Daniels, Werner, and Bahill [2001])

v:	The input value for the FOM
Score:	The output of the scoring function
L:	The lower threshold of the input value

B: Baseline (status quo) value for the FOM. By definition baseline values are always assigned a score of 0.5

S: The slope of the tangent to the scoring function at the baseline value B. This value represents the maximum incremental change in the customer's quantitative judgment with each incremental change in input.

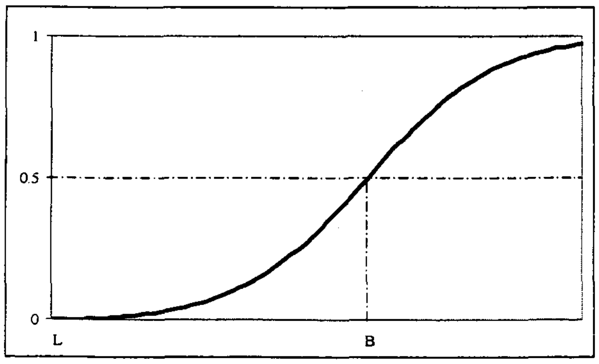
D: The domain of definition of the scoring function.

Of interest here is SSF1 and SSF7, shown in Figure 3. SSF1 is used where high input values are preferable and SSF7 is used where low input values are preferable. Wymore [1993: 390-391] defines these functions as follows

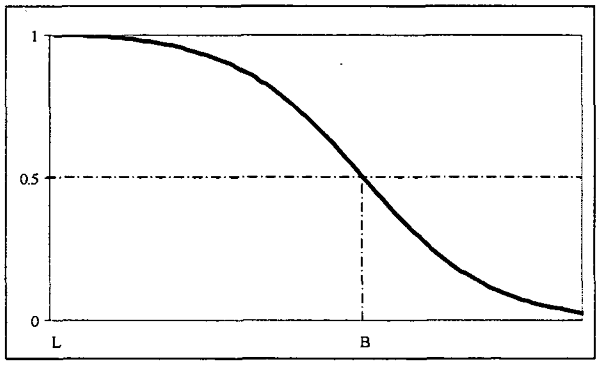
$$\text{SSF1(L,B,S,D)} \quad Score = \frac{1}{1+((B-L)/(v-L))^{2*S(B+v-2*L)}}$$

$$\text{SSF7(L,B,S,D)} \quad Score = 1-\frac{1}{1+((B-L)/(v-L))^{-2*S(B+v-2*L)}}$$

where *v*, *L*, *B*, *S*, and *D* are as defined in the nomenclature above. Now, turning our attention to Figure 3, we point out a few important aspects of the Wymorian standard scoring functions. First, the output of the function is most responsive to change when the input is near the baseline value. Values of *S* and *B* should be chosen such that the input values the customer is most interested in lie within the linear/near-linear portions of the curve. This curve flattens out at the extremes, indicating that changes in these regions are not as important to the customer. Take, for example, a power budget (applying SSF1) where *B* is the baseline required power. When comparing two systems, one with a 10% margin and another with a 50% margin, the customer would probably prefer the one with the 50% power margin. Because these values are near the baseline, the system with the larger margin would receive a notably higher score. Now consider two other systems, one with a 510% margin and one with a 550% margin. These margins are so large to begin with that the small difference between the two is probably of little interest to the customer. Because these values lie far away from the baseline, their scores will be almost identical.



a) SSF1: Large input value is desirable



b) SSF7: Small input value is desirable

Figure 3. Graphs of Wymorian standard scoring functions SSF1 and SSF7.

3.2 Creating Aggregate Figures of Merit

A simple linear combination, shown below, is used to calculate the total evolvability, *f*, of a system from the four different classes of evolvability.

$$f = \sum_{i=1}^{n} w_i x_i$$

Here, *n* is the total number of FOMs to be combined, w_i represents the normalized weight, and x_i represents the score of the i^{th} FOM [Daniels, Werner, and Bahill, 2001]. These normalized weights must be selected such that they reflect the relative value of traversing from the bottom to top of each bottom-level FOM score. These weights may be generated in two ways: direct and indirect. Direct weighting techniques ask the customer to directly select the weights, while indirect weighting techniques ask for interval judgments and derive the weights. Indirect weighting techniques usually provide better results than direct techniques. The most common indirect weighting technique is the paired comparison approach [Buede, 2000]. This technique includes methods such as the analytic hierarchy process (AHP) [Saaty, 1980], trade offs [Watson and Buede, 1987], and the balance beam approach [Watson and Buede, 1987].

4 QUANTIFYING EVOLVABILITY- A THEORETICAL PERSPECTIVE

Evolvability may be measured as a FOM. It is easier to assess a quality like evolvability qualitatively than quantitatively. But as we discuss above, a good FOM must be quantitative. It is proposed that evolvability can be quantitatively modeled through ontology, system complexity, system reconfigurability, and functional isolation. In the following sections we discuss how the characteristics of an evolvable system can be folded into these four metrics.

4.1 A Formal Definition of Ontology

The ontological approach used to model evolvability is based primarily on the work of Mario Bunge [1977 & 1979]. Although philosophical in nature, Bunge's work, which is mathematically exact and rigorous, has lead to extensive research in ontology's application to the engineering and computer science disciplines by authors such as Wand and Weber [1990], Wand and Woo [1991], Bonfatti and Pazzi [1991], and Rowe and Leaney [1997]. What follows in this section is a compilation of the work on ontology done by these authors, followed by an adaptation and expansion of their findings to the concept of evolvability.

4.1.1 Definition and Model of a 'Thing'

Bunge asserts that the world is constructed of entities or substantial individuals called 'things.' Each substantial individual, x, possesses its own unique set of (substantial) properties, $p(x)$. It is, however, "impossible to define an entity as the set of its properties [...] although usually a proper subset of $p(x)$ will suffice to distinguish x from other entities, nothing short of the totality $p(x)$ of properties of x will constitute and individuate it, i.e. render it ontically distinct from every other entity" [Bunge, 1977: 111]. Furthermore, a thing may not be separated from its properties.

A thing may either be simple (consisting of only one thing) or composite (consisting of two or more things). Any two things can be combined into a composite thing. Furthermore, every composite thing must have emergent properties, i.e. properties arising from the combination of things which cannot be directly traced to a property of one of the composing things. Evolvability is an example of an emergent property [Klir, 1985].

The architecture in which we desire to measure and quantify evolvability may be modeled as a special type of composite thing- a system. A system is defined as a composite thing that may not be broken down into non-interacting components. As humans, we create a model of a thing by assigning attributes. Some attributes represent substantial properties (i.e. mass and color), others represent multiple properties (i.e. complexity measures), and still others represent no properties (i.e. a thing's name).

Bunge suggests that a thing, X, may be modeled by a functional schema $X_m = <M, \mathbf{F}>$ where M is a specific reference frame and $\mathbf{F}$ is the state vector (or total state function) of X. The state vector $\mathbf{F}$ is given by

$$\mathbf{F} = <F_1, F_2, \ldots, F_n> : M \rightarrow V_1 \times V_2 \times \ldots \times V_n$$

where F_i is the i^{th} state variable (or state function) and V_i is the domain of possible values for that variable. The concept of ontologically defining the state of a thing is fundamentally similar to describing a model's state when creating a formal model [Zeigler, 1976]. As in system dynamics and control theory [Nise, 1994], a state variable can be defined as the smallest set of linearly independent attributes (system variables) such that the values of the members of the set, along with external and environmental influences, completely determine the value of all the attributes (system variables). Furthermore, a state vector is described as a vector whose elements are the state variables. In other words, the state vector will consist of enough state variables (representing substantial properties) of the proper kind to distinguish one thing from all other things, as discussed above.

All the possible states a thing (or system), X, can achieve is represented by the Cartesian product of the ranges of the state variables. This space, called the possible state space, S(X), is given by

$$S(X) = \{<x_1, x_2, \ldots, x_n> \in V_1 \times V_2 \times \ldots \times V_n \mid x_i = F_i(M) \}$$

It is clear, however, that an actual thing (or system) cannot achieve every conceivable state. Instead it can achieve a subset of these states. This lawful state space, $S_L(X)$, is the portion of the state space a thing can occupy given the appropriate interdependencies of properties, physical limitations, and other constraints.

4.1.2 Modeling Evolvability as an Event

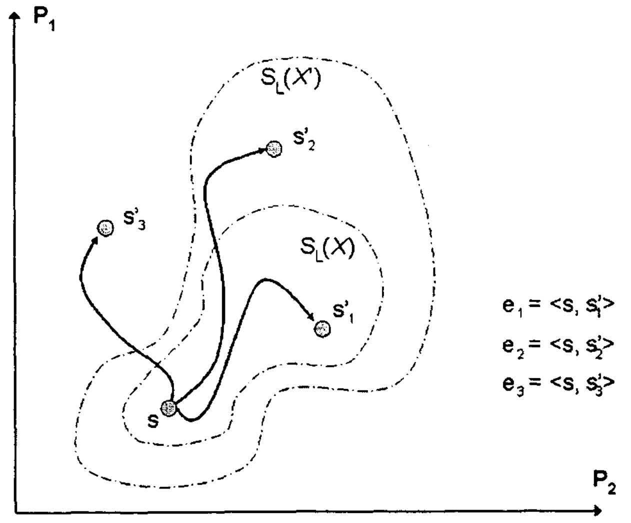
Because evolvability is the ability of a system to accept a change in requirements (resulting in a change of the system state) on a top level it could be viewed simply as the size of the lawful state space. After all, a larger lawful state space represents a greater capacity to accept change. It is conceivable that one could compare the evolvability of multiple systems on this metric alone. Unfortunately, this comparison would be inadequate because some systems may be very evolvable in one direction and not evolvable in another. If the direction of required evolution lies in a non-evolvable direction, then the system can not accommodate this change regardless of the size of the lawful state space. Furthermore, as is discussed later in more detail, the size of the lawful state space of a system only addresses generality (static evolvability) and not the other three classes of evolvability (dynamic evolvability). We will, instead, model evolution as an event.

Evolution is a type of change, and change may be modeled as an event. An event, e, is defined as an ordered pair of states, with each state (beginning, s, and end, s') existing in the possible state space. Therefore, an event is represented mathematically by the following expression:

$$e = \langle s, s' \rangle \quad \text{where } s, s' \in S(X)$$

All events, however, are not possible. If the end state, s', is outside of the lawful state space, $S_L(X)$, then this change may not occur unless the lawful state space is modified to include s'.

The concept of modeling evolvability as an event implies that you must first have knowledge of both the initial and evolved states. Recall that evolvability is the ability to adapt to new requirements. We, therefore, proceed by first describing the original requirements as the initial state, s, followed by defining the evolved requirements as final state, s'. A system may adapt to these changes in requirements statically (through generality) or dynamically (through adaptability, scalability, or extensibility). Figure 4 shows the possible modes of evolution. The first is static evolvability, e_1: the evolved state lies within the lawful state space of the original system, $S_L(X)$. The second is dynamic evolvability, e_2: the evolved state does not lie within the lawful state space of the original system, but the lawful state space may be modified, $S_L(X')$, to include the evolved state. The third is a non-evolvable event, e_3: the evolved state does not lie within the lawful state space of the original system and the lawful state space may not be modified to include the evolved state; this event is not possible.



P_1
$S_L(X')$
s'_2
s'_3
$S_L(X)$
s'_1
s
$e_1 = \langle s, s'_1 \rangle$
$e_2 = \langle s, s'_2 \rangle$
$e_3 = \langle s, s'_3 \rangle$
P_2

Figure 4. Ontological representation of evolvability as an event. Event e_1 represents static evolvability, e_2 represents dynamic evolvability, and e_3 represents a non-evolvable event.

Take, for example, the Boeing F/A-18A Hornet and F/A-18E Super Hornet aircraft [Jackson, 1999]. The lawful state space for each version of the aircraft represents the performance envelope for that aircraft; the increased performance available with the Super Hornet is reflected through a larger lawful state space. Suppose a specific mission requires an unrefuelled combat endurance of two hours. This mission lies outside the lawful state space of the F/A-18A Hornet (with a combat endurance of 1 hr 45 min), but is within the lawful state space of the F/A-18E Super Hornet (combat endurance of 2 hrs 15 min). This is an example of dynamic evolvability: the lawful state space of the system was altered in such a way that includes the new requirements.

4.2 A Discussion on the Role of Complexity

Earlier we argue that evolvability is an emergent property, leading to the conclusion that simple things are not evolvable. This, in turn, suggests that evolvability is only a property of composite things. Furthermore, a system (which is a type of composite thing) may, or may not, possess the emergent property of evolvability.

4.2.1 Is Complexity Good or Bad?

Many engineers are wary of complexity (with good reason) and adhere to Ockham's Razor. This heuristic, first employed by William of Ockham (arguably the most influential philosopher of the fourteenth century) [Moody, 1975], suggests a preference for seeking the simplest solution: "frustra fit per plura quod potest fieri per" (a variety other similar Latin phrases exist in the literature). This may be translated as "it is useless to achieve by more things what can equally well be achieved by fewer" [Tachau, 1988: 132]. We should remember, however, that Ockham's Razor is only a heuristic and not jump to the conclusion that all complexity is bad. Indeed, Klir [1985: 326] states that when striving to obtain certain emergent properties such as evolvability "we search, within given constraints, for systems with a high degree of complexity [...] In some situations, a certain degree of complexity is a necessary condition for obtaining some specific system properties." While some systems require complexity to develop important emergent properties, others require complexity due to the nature of their objectives. These engineering systems are inherently complex because the nature of their mission is itself highly complex [Bar-Yam, 2003]. Such is the case in space systems, air traffic control, and nuclear power plants. Therefore, we argue that in some systems complexity is both advantageous and necessary.

It is important to note, however, that we do not search for needless complexity. After all, "needless complexity is one design dimension that every product developer must attempt to minimize" [Meyer and Lehnerd, 1997: 97]. Needlessly increasing complexity only makes it harder for the engineer to fully understand the system, leading to unforeseen problems and increased failures. Rechtin [1991: 182] suggests that it is "common sense to keep the conceptual model as simple as possible while still satisfying the client's needs [...] Complexity is a breeding ground for errors." For example, components that provide sufficient reliability in simple systems may not provide acceptable reliability in complex systems where they occur more frequently or are used more often. Furthermore, increased complexity usually means increased cost in manufacturing and assembly. Meyer and Lehnerd [1997: 99] state that "reducing complexity almost always reduces direct and indirect costs. Complexity fuels the costs, which grow geometrically if not exponentially." Additional complexity, where it is not necessary, only serves to increase production cost and decrease reliability.

4.2.2 What is Complexity?

Now we turn our attention to developing a more precise definition of complexity and establishing the difference between a complex and complicated system [Waldrop, 1992; Ottino, 2004]. Both complex and complicated systems are ones that have a large number of components interacting in a large number ways. The difference between the two lies in the way these components interact. In a complicated system, a failure or a change in operating environment can bring down an entire system. Examples of complicated systems include elaborate mechanical watches and snow flakes. A complex system, on the other hand, can adapt to a component failure or a change in operating environment. They are complex, adaptive, self-organizing, and usually have a number of autonomous components. Examples of complex systems include highways, the U.S. power grid, the internet, and metabolic pathways [Ottino, 2004]. Furthermore, complex systems exhibit a kind of dynamism, which is not possessed by simply complicated systems, that allows them to exist in a place where the components of the system never quite lock into place, yet never quite dissolve into turbulence [Waldrop, 1992]. This place is known as the 'edge of chaos.'

It is crucial to recognize that engineering systems extend beyond just hardware and software. In many instances, humans are a critical part of the system [Bar-Yam, 2003]. Take, for example, the Space Shuttle- arguably one of the most complicated systems ever built. Despite its many redundant systems, a failure on orbit can have catastrophic consequences. After 20 years of service, the shuttle remains operational by adapting to new requirements and new mission objectives. While the hardware and software do not adapt on their own, the men and women of the shuttle program, as part of the overall system, supply this capability. They are able to leverage the fact that the shuttle has many components interacting in many ways to dynamically address unforeseen problems while on the ground or in orbit. The integration of these innovative personnel into the operation of the system transforms a complicated system into a complex one.

4.2.3 Why Evolution Requires Complexity

Recall that evolvability is an emergent property and is therefore only found in composite things, meaning that some amount of complexity is inherently necessary to make evolution possible in the first place. We continue with the argument that evolvability demands complexity by suggesting that adding additional capabilities or adapting to a new environment increases the complexity of a system [Percivall, 1994]. Furthermore, as a system evolves to meet new requirements it is not only pulled to the edge of chaos, but also pushed along the edge of chaos in the direction of increasing complexity [Waldrop, 1992]. Indeed, Waldrop [1992: 295] points out "the deceptively simple fact that evolution is constantly coming up with things that are more complicated, more sophisticated, more structured than the ones that came before." The notion of increasing complexity as requirements evolve is especially important in a program relying on the spiral development process. This implies that each successive spiral usually results in a more complicated system.

We also approach this from a common sense perspective. The more components a system has the more options there are to reconfigure that system. Furthermore, the more interfaces a system has, the more "hooks" there are into the system. Take, for example, a system with only two components. Generally speaking, there are very few ways these two components may be reorganized or that new components may be added. Now, compare this to a system with 50 components. Clearly the larger, more complex system has more options in how it may be organized and where new components may be added.

4.2.4 Measuring System Complexity

Many ways have been proposed to measure complexity and three of them are addressed here. Specifically, measures of complexity discussed are based on (1) the complexity of the task at hand, (2) the complexity of the assembly, and (3) the amount of information necessary to describe the state of the system.

The first method of measuring complexity is by assessing the complexity of the task to be performed. This measure is based on Ashby's [1956] Law of Requisite Variety. Law of Requisite Variety, as described by Ashby [1956: 207], states that "the variety in the outcomes, if minimal, can be decrease further only by a corresponding increase in that of R [...] To put it more picturesquely: only variety in R can force down the variety due to D; only variety can destroy variety." Here, *R* represents the regulator (or system) and *D* may represent some form of disturbance. Furthermore, to be successful, the variety (complexity) of a system, at a minimum, must match the variety (complexity) of the environment to which it is responding.

The second method for measuring complexity is by assessing the complexity of the assembly. According to general system theory, three types of distinctions may be made between components when dealing with a system [Bertalanffy, 1968]. Components can be described based on their number, species, and relation. Taking this one step further suggests that complexity may be measured by these same characteristics. The Boothroyd-Dewhurst complexity factor [Meyer and Lehnerd, 1997], developed for design for manufacture and assembly (DFMA), measures assembly complexity based on component number, species, and relation. This complexity factor, C_f, is as follows:

$$C_f = (N_p + N_t + N_i)^{1/3}$$

where N_p is the total number of parts, N_t is the number of types (species) of parts, and N_i is the number of interfaces.

The third method for measuring complexity is by assessing the amount of information required to define the state of the system. There are many methods available to measure this, but the method discussed here is functional entropy as described by Min and Chang [1991]. Here, it is argued that the amount of information required to define the state of the system "can be used as a measure of the system complexity, and it can be denoted as the entropy of the system." They go on to assert that "the complexity of a system arises not only due to the number of components and the characteristics of each component, but also to the types of interconnections between the components." A detailed discussion on the calculation of functional entropy, H_F, is provided by Min and Chang [1991].

4.3 The Advantage of Reconfigurable Systems

The ability of a system to reconfigure itself (defined as adaptability) is one of the four fundamental ways a system may evolve to meet new challenges. As discussed above, reconfiguration is one method for altering the lawful state space of a system. In the sections that follow we discuss reconfigurability in more detail.

4.3.1 Reconfigurable Systems

One of the fundamental keys to system reconfigurability is modularity. IEEE Std. 610.12 [1990] defines modularity as "the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components." Modularity is advantageous when dealing with complex systems because it allows the engineer to isolate much of the system's complexity within different modules. These modules can then interact with the rest of the system through simple interfaces. Modularity also allows for modules to be rearranged to optimize the system for a given task or to adapt to achieve new objectives. A more complete discussion of modularity is provided by Baldwin and Clark [1999].

Modularity, however, does little good (from a standpoint of reconfigurability) unless the system also has standard interfaces. Regardless of how the system is modularized, there is little advantage if components only fit together in one way due to unique interfaces. Therefore, equally as important as modularity is the standardization of interfaces. In addition, given the same number of modules, an increase in the number of available interfaces significantly increases the reconfigurability of a system. Simple, standard, and available interfaces are central to enabling the reconfigurability and evolvability of a system. Rechtin [1991: 29] suggests that "the greatest leverage in systems architecting is at the interfaces."

Finally, a reconfigurable system is not achieved for free. Reconfigurability often comes at the cost of other performance metrics. In next generation computer systems, for example, Verbauwhede and Chang [2002] describe the tradeoff between power and reconfigurability. Other similar tradeoffs may exist in different systems; such tradeoffs might include mass-reconfigurability, complexity-reconfigurability, and reliability-reconfigurability.

4.3.2 Example: Modular Reconfigurable (MR) Robots in Space Applications

One example of the advantages gained by using a reconfigurable system can be seen in the use of MR robots in space applications as described by Yim et al. [2003]. Here, it is described how MR robots provide, among other things, adaptability and redundancy. They discuss the use of PolyBot, a MR robot system consisting of two types of modules, to demonstrate how one system may be used to perform the following space missions: space manipulation, surface mobility, and digging. Redundancy is achieved through having many repeated modules. It should be noted, however, that as the number of modules (and hence redundancy) increase the probability of a module failing also increases. This is an example of one of the tradeoffs discussed above.

4.3.3 Example: Distributed Satellite Systems and the TechSat 21 Program

Distributed satellite systems are another example of reconfigurable systems. These systems distribute the functionality of the system between multiple satellites in a satellite cluster. The satellites in this cluster communicate and share the burden of different mission functions. Together they act as one unified system. In many cases, such a system is more attractive than a monolithic design because neither the geometry or the number of satellites in the system is fixed. This is especially valuable in high cost, high value missions where the system's capability is slowly increased over time [Martin and Stallard, 1999].

An excellent example of a distributed satellite system is the TechSat 21 program, described in detail by Martin and Stallard [1999]. This program took advantage of the distributed satellite system approach to synthesize a very large aperture for space based radar. Saleh, Hastings, and Newman [2001] discuss how the TechSat 21 satellite cluster might be reconfigured to operate in a geo-location mode, instead of the baseline radar mode.

4.3.4 Measuring Reconfigurability

Farritor [1998] discusses a method for enumerating the modular design space of a robotic system. In essence, this method identifies the number of ways modules in a modular robot may be reconfigured. Here, we propose to use the foundations of this metric as a method for measuring reconfigurability between multiple systems in a relative fashion. The system that may be reconfigured in the largest number of ways, based on this metric, will be deemed the most reconfigurable.

In his assessment of the modular design space, Farritor introduces the quantity D_{limbs} which represents the number of limbs that can be created from an existing suite of modules and end effectors. The example provided here is for a system with two types modules, *A* and *B*, and one type of end effector, *F*. This model, of course, could be modified to include *n* types of modules. Given N_A modules of type *A* and N_B modules of type *B*, it can be shown that the total number of limb assemblies possible is given by

$$D_{limbs} = \sum_{j=0}^{N_A} \sum_{k=0}^{N_B} \frac{(j+k+(i-1))!}{j!\,k!\,(i-1)!}$$

where *j* is the number of modules of type *A*, *k* is the number of modules of type *B*, and *i* is the number of limbs used. In this case, the number of limbs used, *i*, is the same as the number of end effectors because each limb must end with an end effector. After this, Farritor continues by discussing the contribution of the number of ports available on a central power/control module to which all the limbs are mounted. This contribution, however, does not apply to the type of systems that will be considered in the following sections.

4.4 The Necessity of Functional Isolation, Open Standards, and Interoperability

Functional isolation is another key to system evolvability. An evolvable system should be built on the aspects of the system which are most likely to remain unchanged [Isaac and McConaughy, 1994]. These "islands of architectural stability" [Percivall, 1994] enable a complex system to evolve much quicker and easier than what might otherwise be possible. Indeed, Rechtin [1991] states that "complex systems will develop within an overall architecture much more rapidly if there are stable intermediate forms than if there are not." These islands of stability should be identified by the system architect as early in the development process as possible. As with reconfigurability, islands of architectural stability suggest a modular approach. Such an approach allows the engineer to isolate core functionality in a set of stable, autonomous modules.

Such functional isolation, however, implies that focus should be placed on the architecture level. Here we introduce the concept of an Open Architecture as described by Buede [2000]. An open architecture is one where the hardware and software interfaces are defined such that additional resources can be added to the system with little or no adjustment. Two important aspects for an open architecture when seeking an evolvable system include openness and interoperability. The SMC Systems Engineering Primer and Handbook [2004] defines these properties as follows:

Open Standards: Parts, modules, objects, products and systems are based on vendor-independent, non-proprietary, publicly available, and widely accepted standards. Standards allow for a transparent environment where users can intermix hardware, software, and networks of different vintages from different vendors to meet different needs.

Interoperability: The ability of systems, units, or forces to provide and receive services from other systems, units, or forces and to use the services so interchanged to enable them to operate efficiently together.

The importance of open standards and interoperability should be self-evident. They ensure that all hardware and software will interface in the event that evolving requirements or other circumstances require the components to be reconfigured or otherwise adapted. Using Apollo XIII as an example, open standards and interoperability ensure that there are not square CO_2 filters on the command module and round filters on the lander (you don't find yourself making "a square peg fit into a round hole").

5 CASE STUDY- EVOLVING LUNAR MISSIONS TO MARS MISSIONS

We now transition to a more practical perspective and address how the ideas proposed above may be applied to quantifying evolvability in real-world problems. Recent events have turned the eyes of the United States back to the moon, this time as a stepping stone to missions beyond. In his speech at NASA Headquarters in Washington, D.C. on January 14, 2004, President George W. Bush said that "returning to the moon is an important step for our space program. Establishing an extended human presence on the moon could vastly reduce the costs of further space exploration, making possible ever more ambitious missions." He went on to say that "with the experience and knowledge gained on the moon, we will then be ready to take the next steps of space exploration: human missions to Mars and to worlds beyond." It is clear, therefore, that the systems developed to accomplish these near term lunar missions should lend themselves to adaptation in order to meet the requirements imposed by Mars missions. These systems must be evolvable. Indeed, the President's Commission on Implementation of United States Space Exploration Policy [Aldridge et al., 2004: 27] recommends that NASA "design an exploration architecture that evolves iteratively, systematically through a series of so-called spiral developments." Here we apply the quantitative approaches to assessing evolvability discussed in the previous sections.

Before proceeding, a few words about the models presented in this case study. When generating a model, it is always important to remember that "a model is not reality" [Rechtin, 1991: 58]. As such, a model should not be taken at face value and should always be subject to review and improvement. The models proposed here are early examples of how the ideas introduced above may be applied. Much of the author's future work involves readdressing and refining these preliminary models.

5.1 Generality

As the only static mode of evolvability, generality is probably the most straightforward of the four classes of evolvability. Generality may be viewed as the "do-nothing" alternative. If the evolved state (Mars mission) lies within the lawful state space of the original architecture (lunar mission), the architecture is assigned a generality score of one, otherwise it is given a score of zero.

To make this assessment, we must first address what state variables define the state space of this architecture. For this particular case study, potential state variables include delta-V [Hale, 1994], habitable volume during different mission phases [Fraser, 1966], power system requirements [Landis, McKissock, and Bailey, 1999], and propulsion system selection [Hurlbert, 1993] to name a few.

5.2 Adaptability

Adaptability modifies the lawful state space of the architecture. Unfortunately, it is difficult to say exactly how the lawful state space will be modified by reconfiguring elements in a potentially unknown way. Therefore, we propose to measure adaptability simply as a function of how reconfigurable the system is, without consideration of the modified shape or size of the lawful state space.

To do this, we first turn to Farritor's measure of reconfigurability introduced in a previous section. Recasting this measure as it applies to the case study at hand, we generate the following expression

$$D_{reconfig} = \sum_{a=0}^{N_A} \sum_{b=0}^{N_B} \cdots \sum_{n=0}^{N_N} \frac{(a+b+\cdots+n+(i-1))!}{a!b!\cdots n!(i-1)!}$$

where $D_{reconfig}$ is the measure reconfigurability, N_A is the number of modules of type A, N_B is the number of modules of type B, and so on. The number of propulsion modules is represented by i. Notice that in this instance, propulsion modules take the place of Farritor's end effectors. This can be done because it is assumed that every configuration of vehicles in a lunar/Mars architecture must terminate in a vehicle with propulsive capability (otherwise the stack of vehicles would just sit in Low Earth Orbit). Furthermore, recall that this system must be modular and have standard interfaces to be reconfigurable. Therefore we define the reconfigurability score, F_{adapt}, as follows

$$F_{adapt} = (N_{Modular})(N_{StdInterfaces})(D_{reconfig})$$

where $N_{Modular}$ is one if the architecture is modular, zero otherwise; and $N_{StdInterfaces}$ is one if the architecture has standard interfaces, zero otherwise. By definition, therefore, if the architecture is not modular with standard interfaces, the reconfigurability score is automatically zero.

Clearly, a large reconfigurability score is desirable. Now applying this metric to a Wymorian standard scoring function (SSF1), we obtain the following expression for the adaptability score

$$Adaptability = \frac{1}{1+((B-L)/(F_{adapt}-L))^{2*S(B+F_{adapt}-2*L)}}$$

Now we must define the quantities B, S, and L. For this application, for example, we may assume a baseline of $B = 5$, representing two modules and one propulsion module (i.e. Apollo- lander, command module, and service module); a lower bound of $L = 0$, representing no modules; and a slope of $S \approx 0.2$.

In some instances, Farritor's measure of reconfigurability may be too complicated to easily calculate. In these cases, the Boothroyd-Dewhurst complexity factor may be substituted as an approximate measure of reconfigurability. The other assumptions of modularity and standard interfaces must, of course, remain the same. Under these conditions a change to the architecture (increase/decrease in modules or interfaces) causes the same trend in Farritor's measure of reconfigurability and the Boothroyd-Dewhurst complexity factor (they both increase or both decrease). Although the same trends are evident, they do not respond in the same way to identical changes. For example, adding a module has a much greater impact on the reconfigurability measure than on the complexity measure.

5.3 Scalability

Scalability also modifies the lawful state space of the architecture, but in a way much more measurable than in adaptability. We begin by assessing how much the lawful state space must stretch to engulf the evolved state. The distance from the edge of the original lawful state space to the evolved state, along the direction of the i^{th} state variable, is defined as d_i. If a state variables in the evolved state lies within the original lawful state space, $d_i = 0$.

Next we assess the difficulty of stretching the state space along the direction of the i^{th} state variable by a distance d_i. This may be done in a number of ways. The customer, for example, may be allowed to pick the difficulty (perhaps on a scale of 1 to 5). This, however, is not the recommended approach. Instead difficulty should be assigned based on the properties of that particular state variable. Power, for example, could be assigned a difficulty factor based on the difficulty of increasing the size of the solar arrays or increasing the size of the reactant tanks for fuel cells. Or, for the propulsion system, difficulty may be assigned based on properties such as propellant selection, if the system is pressure-fed or pump-fed, if the system is liquid or gaseous, or if the OMS and RCS systems are integrated. Regardless of the methods or properties chosen, this difficulty factor, z_i, should be carefully developed with input from the customer.

To each distance, d_i, we also apply a weighting factor, w_i. This weighting factor addresses the scale of that particular state variable. This prevents a variable such as delta-V (with magnitudes in the hundreds and thousands of m/s) from overpowering something like crew size (on the order of 5 to 10 people). Given this information, we define the weighted scaling distance, F_{scale}, as

$$F_{scale} = \sqrt{\sum_{i=1}^{n} w_i^2 z_i^2 d_i^2}$$

where n is the number of state variables. Here, it is clear that a lower F_{scale} is desirable. Indeed, if $F_{scale} = 0$ then the evolved state is included in the original lawful state space and no scaling is necessary. This is the most desirable state and should result in a scalability score of one (it also results in a generality score of one). Therefore, we use SSF7 and obtain the following expression for the scalability score:

$$Scalability = 1 - \frac{1}{1 + \left((B - L)/(F_{scale} - L)\right)^{-2*S(B+F_{scale}-2*L)}}$$

Defining the quantities B, S, and L for this application, we will assume a lower bound of $L = 0$. The values of B and S may be determined based on the state variables chosen to represent the system.

5.4 Extensibility

As with the other forms of dynamic evolvability, extensibility modifies the lawful state space. Furthermore, like adaptability, there is no good way to measure how extensibility may alter the lawful state space. After all, in extensibility we are changing the state space by adding an unforeseen number of components, of an unknown type, with unknown properties. Because of these unknowns, much work remains to be done before such a relationship can be suggested with sufficient confidence. A relationship for extensibility would likely be structured in the same manner as adaptability and scalability. First, a measure of extensibility, F_{extend}, would be generated. Then, this metric would become the input to a Wymorian standard scoring function (SSF1 or SSF7 depending on if a high or low value of F_{extend} is desired). The output of this scoring function would become the extensibility score.

5.5 Evolvability

To calculate the total evolvability score, we use a simple linear combination. The relative weighting of each class of evolvability is up to the customer and weights may be assigned using any of the methods mentioned earlier. We come, therefore, to the following expression for the total evolvability FOM score

$$Evolvability = (A*Generality) + (B*Adaptability) + (C*Scalability) + (D*Extensibility)$$

Furthermore, the evolvability score shows compensation as described by Daniels, Werner and Bahill [2001]. A high adaptability and extensibility score with a low generality and scalability score is, given similar weighting, just as good as a low adaptability and extensibility score with a high generality and scalability score. After all, architectures must not exhibit all classes of evolvability to effectively evolve- excelling in only one class may be all that is necessary.

6 CONCLUSIONS

Current events have made evolvability a sought after attribute. Here we propose measuring evolvability as a FOM. This helps us address the following question: is evolvability really a desirable property? By considering evolvability among a host of other FOMs (such as cost, reliability, performance, etc.) the customer can decide if they can afford a highly evolvable system. As we mention earlier, evolvability is not free. It usually comes at the price of other performance metrics, primarily cost and reliability.

The approach proposed here provides a foundation for further work in quantifying evolvability. Furthermore, the case study presents an early example of how these ideas might be applied to an actual problem. Much work, however, remains in refining these measures, especially the measure of extensibility. With more work, we believe reliable and accurate measures of evolvability may be established.

ACKNOWLEDGEMENTS

The author thanks David Forrest and Erica Walsh of the Exploration Systems Engineering Office at the NASA Johnson Space Center for reviewing the manuscript. The author also appreciates the helpful input of Joyce Carpenter, Jim Geffre, Jon Lenius, and Paul Dum.

REFERENCES

E.C. Aldridge et al., A Journey to Inspire, Innovate, and Discover, President's Commission on Implementation of United States Space Exploration Policy, 2004.

W.R. Ashby, An Introduction to Cybernetics, John Wiley & Sons, Inc., New York, NY, 1956.

C.Y. Baldwin and K.B. Clark, Design Rules Vol. 1: The Power of Modularity, MIT Press, Cambridge, MA, 1999.

Y. Bar-Yam, When Systems Engineering Fails- Toward Complex Systems Engineering, IEEE Proc Syst Man Cybernet, October 5-8, 2003, pp. 2021-2028.

L. von Bertalanffy, General System Theory, George Braziller, Inc., New York, NY, 1968.

B.W. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer 21(5) (1988), 61-72.

F. Bonfatti and L. Pazzi, Modeling Object Complexity and Behavior: Towards an Ontological Paradigm, IEEE Proc CompEuro, May 13-16, 1991, Bologna, Italy, pp. 844-849.

D.M. Buede, The Engineering Design of Systems: Models and Methods, John Wiley & Sons, Inc, New York, NY, 2000.

M. Bunge, Treatise on Basic Philosophy, Volume 3, Ontology I: The Furniture of the World, Reidel, Boston, MA, 1977.

M. Bunge, Treatise on Basic Philosophy, Volume 4, Ontology II: A World of Systems, Reidel, Boston, MA, 1979.

G.W. Bush, "A Renewed Spirit of Discovery," NASA Headquarters, Washington, D.C., January 14, 2004.

J. Daniels, P.W. Werner, and A.T. Bahill, Quantitative Methods for Tradeoff Analyses, Systems Engineering 4(3) (2001), 190-212.

S. Farritor, On Modular Design and Planning for Field Robotic Systems, Ph.D. Thesis, Dept. of M.E, MIT, Cambridge, MA, May, 1998.

T.M. Fraser, The Effects of Confinement as a Factor in Manned Space Flight, NASA/CR-511, 1966.

F.J. Hale, Introduction to Space Flight, Prentice Hall, Upper Saddle River, NJ, 1994.

E. Hurlbert, Lunar Lander and Return Propulsion System Trade Study, NASA/TP-3388, August 1993.

IEEE 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, Institute of Electrical and Electronic Engineers, New York, NY, 1990.

D. Isaac and G. McConaughy, The Role of Architecture and Evolutionary Development in Accommodating Change, Proc 4th Annu Sym Nat Council Syst Eng (NCOSE), August 1994, San Jose, CA, pp. 541-545.

P. Jackson (ed.), Jane's All the World's Aircraft: 1999-2000, Jane's Information Group, 1999.

G.J. Klir, Architecture of Systems Problem Solving, Premium Press, New York, NY, 1985.

G.A. Landis, B.I. McKissock, and S.G. Bailey, "Designing Power Systems," in W.J Larson and L.K. Pranke (Editors), Human Spaceflight Mission Analysis and Design, McGraw-Hill, New York, NY, 1999.

M. Martin and M.J. Stallard, Distributed Satellite Missions and Technologies- The TechSat 21 Program, AIAA-99-4479, Proc AIAA Space Technology Conference and Exposition, September 28-30, 1999, Albuquerque, NM.

M.H. Meyer and A.P. Lehnerd, The Power of Product Platforms: Building Value and Cost Leadership, The Free Press, New York, NY, 1997.

B. Min and S.H. Chang, System Complexity Measure in the Aspect of Operational Difficulty, IEEE Trans Nuclear Science 38(5) (1991), 1035-1039.

E.A. Moody, Studies in Medieval Philosophy, Science, and Logic, University of California Press, Berkeley, CA, 1975.

H. Mooz and K. Forsberg, Clearing the Confusion About Spiral/Evolutionary Development, Proc 14th Annu Sym Int Council Syst Eng (INCOSE), June 20-24, 2004, Toulouse, France.

NASA Systems Engineering Handbook, SP-6105, National Aeronautics and Space Administration, June 1995.

G.A. Newberry, Changes from DOD-STD-2167A to MIL-STD-498, CrossTalk- The Journal of Defense Software Engineering (April 1995).

N.S. Nise, Control Systems Engineering, Third Edition, John Wiley & Sons, Inc., New York, NY, 1994.

J.M. Ottino, Engineering Complex Systems, Nature 427 (2004), 399.

G.S. Percivall, System Architecture for Evolutionary Development, Proc 4th Annu Sym Nat Council Syst Eng (NCOSE), August 1994, San Jose, CA, pp. 571-575.

E. Rechtin, Systems Architecting: Creating and Building Complex Systems, Prentice Hall, Englewood Cliffs, NJ, 1991.

D. Rowe and J. Leaney, Evaluating Evolvability of Computer Based Systems Architectures- An Ontological Approach, IEEE Proc ECBS, March 24-28, 1997, Monteray, CA, pp. 360-367.

D. Rowe, J. Leaney, and D. Lowe, Defining Systems Evolvability- A Taxonomy of Change, IEEE Proc ECBS, March 30-April 3, 1998, Jerusalem, Israel, pp. 45-52.

T.L. Saaty, The Analytic Hierarchy Process, McGraw-Hill, New York, NY, 1980.

J.H. Saleh, D.E. Hastings, D.J. Newman, Extracting the Essence of Flexibility in System Design, IEEE Proc NASA/DoD Workshop on Evolvable Hardware, July 12-14, 2001, Long Beach, CA, pp. 59-72.

SMC Systems Engineering Primer and Handbook, Space and Missiles Center, U.S. Air Force, 2004.

K.H. Tachau, Vision and Certitude in the Age of Ockham, Brill Academic Publishers, Leiden, NY, 1988.

I. Verbauwhede and M.C.F. Chang, Reconfigurable Interconnect for Next Generation Systems, ACM Proc Int Workshop Syst Level Interconnect Prediction (SLIP), April 6-7, 2002, San Diego, CA, pp. 71-74.

M.M. Waldrop, Complexity: The Emerging Science at the Edge of Order and Chaos, Simon & Schuster, New York, NY, 1992.

Y. Wand and R. Weber, An Ontological Model of an Information System, IEEE Trans Software Engineering 16 (11) (1990), 1282-1292.

Y. Wand and C.C. Woo, An Approach to Formalizing Organizational Open System Concepts, ACM Proc Organizational Computing Systems, November 5-8, 1991, Atlanta, GA, pp. 141-146.

S.R. Watson and D.M. Buede, Decision Synthesis: The Principles and Practice of Decision Analysis, Cambridge University Press, Cambridge, UK, 1987.

A.W. Wymore, Model-Based Systems Engineering, CRC Press, Boca Raton, FL, 1993.

M. Yim, K. Roufas, D. Duff, Y. Zhang, and S. Homans, Modular Reconfigurable Robots in Space Applications, Autonomous Robots 14(2-3) (2003), 225-237.

B.P Zeigler, Theory of Modelling and Simulation, Krieger Publishing Co, Malabar, FL, 1976.